Zaliczenie wykładu JiMP2, kolokwium 2

Grupa B 17 stycznia 2007

1. Proszę wyjaśnić czym różni się zgłoszenie wyjątku od wykonania instrukcji `return`. (Obie instrukcje, tzn. `throw` i `return` kończą wykonanie bieżącej metody i obie mogą być wykonane w razie wystąpienia np. nieoczekiwanych (błędnych) danych, ale z punktu widzenia obsługi błędów i z punktu widzenia przepływu sterowania są pomiędzy nimi istotne różnice – proszę je opisać.) [10 pkt.]

2. Proszę opisać mechanizm zgłaszania-obsługi zdarzeń i jego rolę w projektowaniu graficznych interfejsów użytkownika z wykorzystaniem biblioteki Swing. [15 pkt.]

3. Proszę zaprojektować klasę `Circle` (reprezentującą okrąg o promieniu r) implementującą interfejs `ComparableFigure`:

```
public interface ComparableFigure {
   public boolean sameFigure( Object o );
   public int compareTo( Object o );
}
```

   Działanie metod `sameFigure` i `compareTo` powinno polegać na porównywaniu promieni okręgów.
   Poza implementacją `ComparableFigure` klasa `Circle` powinna zawierać przynajmniej jeden konstruktor i metodę `toString`. [20 pkt.]

   Czy możliwa jest deklaracja tablicy w postaci:
   `ComparableFigure [] drawing = new ComparableFigure[100];`
   Do czego taka konstrukcja mogłaby służyć? [5 pkt.]

   Jak zmienić deklarację interfejsu `ComparableFigure`, aby kompilator mógł kontrolować, czy nie próbujemy porównywać różnych figur (tzn. sygnalizował błąd na przykład przy wywołaniu `sameFigure` z argumentem typu `Triangle` dla obiektu typu `Circle`. [10 pkt.]

Zaliczenie wykładu JiMP2, kolokwium 2

Grupa A 17 stycznia 2007

1. Proszę wyjaśnić, czym różni się interfejs od klasy szkieletowej. Najlepiej posłużyć się przykładem interfejsu `MouseListener` i klasy `MouseAdapter` lub podobnej pary z biblioteki Swing. [10 pkt.]

2. Proszę opisać z czego wynika konieczność synchronizacji dostępu do danych w programowaniu wielowątkowym i jakie mechanizmy synchronizacji udostępnia Java. [15 pkt.]

3. Proszę zaprojektować zwykłą klasę `Sphere` rozszerzającą abstrakcyjną klasę `Figure3D`

```
public abstract class Figure3D implements ZetComparable {
   private String typename= “Figure3D”;
   private point3D center;
   private long zIndex;

   abstract public double volume();
   abstract public double surface();
   public void move( Point3D v ) {
     center.add( v );
   }
   public String toString() {
     return “Error: this method should be overwritten!”;
   }

   public long getZIndex() {
     return zIndex;
   }
   public int zetCompare( ZetComparable other ) {
     long otherIndex= other.getZIndex();
     if( zIndex > otherIndex)
       return 1
     else if( zIndex < otherIndex )
       return -1;
     else
       return 0;
   }
}
```

Projekt klasy: [15 pkt.]

Proszę wyjaśnić, na czym polega polimorfizm na przykładzie deklaracji
`Figure3D [] scene = new Figure3D[1000];` [10 pkt.]

Czy poprawna jest poniższa instrukcja? Proszę uzasadnić odpowiedź.
`ZetComparable [] scene= new Figure3D[1000];`
Jak prawdowpodobnie wygląda interfejs `ZetComparable`? [10 pkt.]